The Report committee for Jeremy James Hintz
Certifies that this is the approved version of the following report:

Generative Adversarial Reservoirs for Natural Video Prediction

APPROVED BY

SUPERVISING COMMITTEE:

Chandrajit Bajaj, Supervisor

Philipp Krähenbühl

Generative Adversarial Reservoirs for Natural Video Prediction

by

Jeremy James Hintz, B.S.C.S.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computational Science, Engineering and Mathematics

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2016

Acknowledgments

I would like to thank a few individuals that went above and beyond in helping me achieve my goals:

- My parents for teaching me the values that have made me the person I am today. Their example of hard work and dedication is as constant as it is inspirational.

- Dr. Alison Norman, PhD Computer Science from UT Austin. I served as a TA for Dr. Norman for several semesters. Her mentorship has been invaluable to me in pursuing my academic goals, and I thank her for all that she has done for me.

- Dr. Annette Taberner-Miller, PhD MIT. Once my boss but now counted as a mentor and dear friend, she taught me a lot about machine learning and data science, but even more about how to craft a powerful narrative using data in order to drive an organization forward.

- Jen Quinlan, VP at Rithmio. Jen demonstrated passion for technology and its transformative power in society. Her advocacy and teaching has made a tremendous difference in my early career. I can only hope that I am able to pay forward the kindness and support she has shown me to an aspiring technologist of the next generation.

Generative Adversarial Reservoirs for Natural Video Prediction

Jeremy James Hintz, M.S.C.S.E.M.
The University of Texas at Austin, 2016

Supervisor: Chandrajit Bajaj

In this report, we will give a brief overview of selected deep learning technologies in the interest of developing both understanding and motivation for the use of reservoir computing and generative models. Furthermore, we will show that these concepts can be applied to the problem of natural video prediction. Influenced by [9], [41], and [31], we develop a novel architecture called Generative Adversarial Reservoirs (GAR). We use GARs to predict frames of videos from the UCF-101 dataset [48] and show that although some of the quantitative evaluations for our results are below state-of-the-art, utilizing reservoirs allows our model training to converge significantly faster while still achieving qualitatively good results.

Table of Contents

List of Figures

Chapter 1

Introduction

Deep learning techniques have already been applied to a variety of fields in both academia and industry, attaining state of the art results in computer vision [25, 49, 47], speech recognition [7], automatic game playing [33, 34, 46], and a variety of other noteworthy applications. In the past 2-3 years, natural video prediction has become a prominent area of deep learning research. Video learning is an essential challenge for autonomous agents to interact with the world around them. The generation of a temporally consistent video frame, however, is a problem of high dimension. This makes current techniques for natural video prediction impractical for real-world applications where autonomous robots are involved.

In this report, we will give a brief overview of selected deep learning technologies in the interest of developing both understanding and motivation for the use of reservoir computing and generative models. Furthermore, we will show that these concepts can be applied to the problem of natural video prediction. Influenced by [9], [41], and [31], we develop a novel architecture called Generative Adversarial Reservoirs (GAR). We use GARs to predict frames of videos from the UCF-101 dataset [48] and show that although some of

the quantitative evaluations for our results are below state-of-the-art, utilizing reservoirs allows our model training to converge significantly faster while still achieving qualitatively good results.

Chapter 2

Neural Networks: History and Overview

In this section, we take a ground-up approach to understanding and motivating relevant deep learning concepts by means of an overview of important moments in their history.

2.1 Early Years and Background

Humans have sought to understand the nature and inner-workings of the mind since the time of the ancient Greeks. Aristotle and Plato formalized the foundations of logic and deductive reasoning which would be of paramount importance to the intellectual and scientific advancements of the next thousand years. In addition to the opining done by philosophers on the nature of human intelligence, others have imagined what artificial intelligence could look like. Mythology from as early as 500 B.C.E. tells of Hephaestus, the blacksmith who built mechanical workers to assist him in his metalwork and Pigmalion, the sculptor who fell in love with one of his works come to life.

More modern tales like Mary Shelley's *Frankenstein* (1818) and Capek's *Rossum's Universal Robots* (1921) further explored the implications of man creating artificial life. Indeed even before Alan Turing proposed the first Turing

Machine [51] in 1936, people were already wondering what might happen if a programmable machine were to become intelligent.

From the earliest computers, programmable machines have excelled at tasks that require many computations. Computers can add, multiply, subtract, and divide much more quickly than humans, thus they are well-suited for problems that are solved using formulaic mathematical rules. This penchant for computation made it possible, for example, for IBM's Deep Blue to defeat world champion Garry Kaparov in chess using a combinatorial search through the possible moves, evaluating 200 million positions per second [6]. What was widely hailed as one of the first landmark accomplishments for artificial intelligence actually didn't require much more than heavy computation.

While tasks like chess require much intense thought for even very intelligent humans, most humans can easily distinguish between, for example, images of cats and dogs. Until very recently, however, computer programs struggled to perform these simple tasks. Tasks like describing an image [24] or writing original sentences [23], while simple for humans, pose serious challenges to computers. This disparity provides insight into the motivation for the development of neural networks. The thinking goes that if we could replicate the functions of the brain that cause neurons to fire in response to specific stimuli, perhaps we could achieve results more similar to human performance in tasks similar in nature to those above.

2.2 The Perceptron

There are some who believe that in order to write a computer program that can perform well on tasks that humans perform with ease, we must model that program off the human brain itself. It was largely this thinking that led Frank Rosenblatt to propose the perceptron [42] in 1957. Funded by the U.S. Office of Naval Research, the perceptron modeled a single neuron. Using relatively simple algorithms, it was able to learn a multitude of functions. This simple model initially appeared very promising, with some of the first applications including simple image recognition tasks. After an interview with Rosenblatt, *The New York Times* proclaimed that the perceptron was "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." [37].

The single-layer perceptron is a binary classifier that works as follows to model a neuron: given a a real-valued vector x, the perceptron outputs a value $f(x)$:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0, \\ 0 & \text{otherwise,} \end{cases}$$

where w is a vector of real-valued weights and b is a bias term. Figure 2.1 shows how this process is used to approximate $f(x)$.

2.3 Multi-layer Perceptron and Feedforward Networks

Much of the promise and excitement generated by the perceptron was tempered when Marvin Minsky and Seymour Papert proved that it was impos-

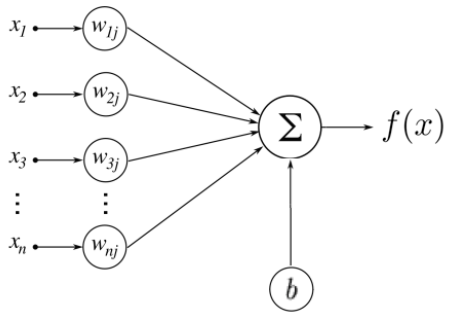

Figure 2.1: The single-layer perceptron models a single neuron, mapping an input vector x to a value $f(x)$ using a weigheted sum with weights w and bias term b.

sible for the perceptron to learn a simple XOR function [32]. The realization that many types of patterns and functions could not be modeled by the perceptron stymied research in the field of neural networks for many years, although it was shown as early as 1972 by Peter Grossberg that multi-layer networks could model many more functions, including XOR [14].

A multilayer perceptron, which came to be known as a feedforward network, is a directed graph where each node is a neuron defined by an activation function and the edges between nodes represent connections for data to flow from one neuron to the next, just as synapses fire between neurons of the human brain.

Specifically, a multi-layer perceptron learns the function $\varphi(\cdot) : \mathbb{R}^n \to \mathbb{R}^k$ where n is the dimension of the input and k is the dimension of the output. The first layer is known as the input layer, and takes a vector $x = x_1, x_2, ..., x_n$ and an objective y. The second, and possibly subsequent layers prior to the

last layer are known as hidden layers. Each of these layers is a set of neurons where each neuron transforms the data from the previous layer using the inner product with a weight vector w. In most networks, this weighted summation is followed by an activation function. The final layer is called the output layer, where the network outputs $f(x)$ to approximate y. Figure 2.2 shows a simple feedforward network as described above.

As we generally seek to learn functions that are not strictly linear, activation functions are generally non-linear. The most common activation function used for most state-of-the-art networks today is the Rectified Linear Unit (ReLU) [8], which is defined as

$$f(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

While other functions, such as $tanh(x)$ and the sigmoid function, are sometimes used, ReLU has been shown to provide good performance with little computational cost [8].

While a multi-layer perceptron is powerful in its univeral approximation ability, there are disadvantages associated with these networks. Notably, hidden layers in a muti-layer network have non-convex loss functions with multiple possible local optima. This means that results can be initialization-dependent.

2.4 Backpropagation

The field of neural networks did not generate much new interest until Paul Werbos introduced the backpropagation algorithm in 1975 [54]. As in

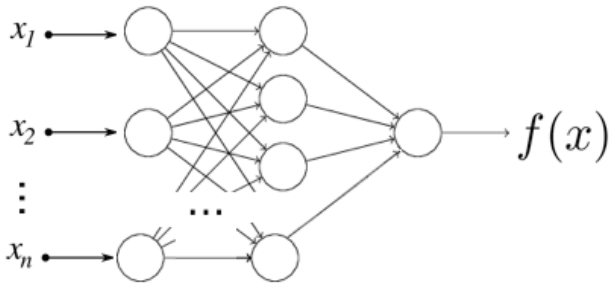

Figure 2.2: The single-layer perceptron models a single neuron, mapping an input vector x to a value $f(x)$ using a weigheted sum with weights w and bias term b.

many forms of classical machine learning, learning occurs through gradient descent. Concretely, the set of weights which minimizes the error between the target and the output of the network is the solution of the learning problem, and we approach this solution by following the gradient at each iteration step.

Backpropagation is an application of the chain-rule applied to a loss function. The steps of the backpropagation algorithm are (1) feeding forward through the network and (2) propagating backward through the network. Consider an input x fed into a network that seeks to approximate an underlying function $g(x)$. As we traverse forward through the network, the primitive functions at the nodes and their gradients are evaluated for each neuron. Conversely, as we traverse backward, we begin by feeding the gradient of the loss function into the output layer. Incoming data to a neuron is summed together and the result is multiplied by the derivative stored in the node from our forward traversal. The result at the input layer is the network derivative $g'(x)$. We use this derivative to update the weights of the neurons in each

layer. The size of the step we take in the direction of the derivative is known as the *learning rate*. Training is stopped when either the target and predicted values are sufficiently close or the difference in weights from one iteration to the next is sufficiently small.

2.5 Connectionism

The renewed popularity of neural networks came to fruition in the early 1980's under the name *connectionism* [43, 44]. Connectionism was largely driven by cognitive science and neuroscience. We originally framed the motivation for neural networks as a method for reverse engineering human intelligence for replicating functional accuracy on specific tasks. The connectionists, on the other hand, believed that, in the same way, human intelligence could also be better understood by implementation [50]. While the endeavor to understand how the brain works on an algorithmic level is still prominent today, it has largely diverged into its own field. The computational neuroscientists of today are researchers that largely wish to understand human cognition through the lens of neural implementation. This knowledge of two fields, however, often empowers these researchers to contribute to both fields. In 1980, Kunihiko Fukushima proposed the Neocognitron, a network architecture for processing images [10]. The architecture was based on the visual architecture of mammals and centered on the notion that computational units become intelligent through interactions with other units, just as it is in the brain.

In the 1990's, many other forms of machine learning came to the fore-

front. While "classical" machine learning techniques where hand-picked and engineered features are used produced state-of-the-art results, neural network research continued, albeit largely in the background. Among the most significant findings of the mid- to late- 1990's include the use of LSTM for addressing inherent mathematical difficulties in modeling long sequences using neural networks [18].

2.6 Deep Learning

In 2006, The trajectory of neural network research changed when Geoffrey Hinton used greedy layer-wise pre-training [17] proving deep belief networks. As others built off of this finding and began to show how many forms of deep networks could achieve promising results using Hinton's strategy, the term "deep learning" began to enter the academic lexicon. Throughout the early 2000's, a growing group of researchers, perhaps most notably Yann Lecun [28, 26, 27], argued vocally in favor of the use of the deep learning paradigm due to its ability to generalize effectively from relatively small datasets.

It wasn't until 2012 that deep learning really took off when Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton won the ImageNet Large-Scale Visual Recognition Challenge. Their network, dubbed AlexNet [25] was a "large, deep convolutional network" that achieved the best top-5 test error rate, coming in at 15.4%, significantly ahead of the 26.2% achieved by the second place team. The success of AlexNet marked a significant turning point in the field of computer vision, and is widely considered one of the most influential machine

learning publications in the field's history.

From 2012 onward, the use of neural networks, especially deep convolutional neural networks (CNNs) became extremely common. Such networks taking home best prize honors at top conferences became increasingly common as well. At this point, the deep learning paradigm is well-known outside machine learning and AI communities and is being used in many major tech companies around the world in addition to being applied to many other scientific fields.

Two of the developments which most positively influenced the success of deep networks were an increase in availability of data and an increase in computing power, most notably through the use of high-powered GPUs. As of writing, the rule of thumb is that "a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category, and will match or exceed human performance when trained with a dataset containing at least 10 million labeled examples" [3]. Similarly, faster processing has allowed researchers to build larger networks, which have been able to generalize better for a variety of tasks. Given the current rate of network growth as facilitated by computing power, CNNs will have the same number of neurons as the human brain by 2050 [3]. It is reasonable to expect that, based on performance of past models, performance will continue to improve as the amount of available data grows and processing speeds improve.

Opposite all the tremendous advantages that deep learning has brought about stands a stark reality: even on the fastest available GPUs, training a

deep network takes a very long time. In the coming chapters, we will address ways to cut down on training time, most notably through reservoir computing.

Chapter 3

Reservoir Computing

One of the solutions to a In 2001, Wolfgang Maass proposed liquid state machines. Not long after, Herbert Jaeger proposed echo state networks. The two concepts are now often grouped together and referred to as reservoir computing. As opposed to the well-defined structure of feedforward networks, reservoirs are usually populated at random. In this section, we will build up some of the pre-requisities for understanding the reservoir computing paradigm and show how it can be used to learn functions in a fraction of the time.

3.1 Recurrent Neural Networks

A neural network that contains cycles is called a recurrent neural network (RNN). We showed that feedforward networks could approximate analytical functions. RNNs, on the other hand, approximate dynamical systems, or functions with a time component. The concept of RNNs has been proposed independently multiple times by different researchers from cognitive science, computational neuroscience, and computer science [11, 45, 52].

In essence, RNNs are no different than feedforward networks, save that we introduce a new, special type of edge that is different from conventional

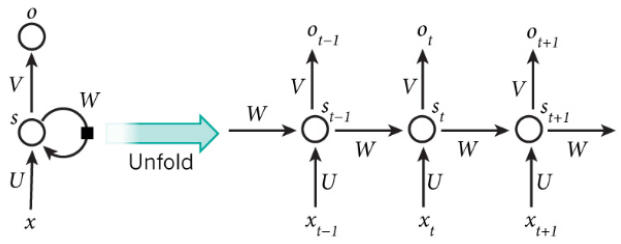

Figure 3.1: A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Image taken from [27].

edges in that it can span adjacent time steps. These edges are called recurrent edges, and they are the type that may form cycles of arbitrary length. Given an input x_t, where t is the current time, and the previous state information, r_{t-1}, We wish to predict an output y_t. We must first calculate the new state r_t, which we do as follows:

$$r_t = \varphi(w \cdot x_t + w^* \cdot r_{t-1} + b_r)$$

where w and w^* are the set of conventional weigths between neurons and recurrent weights between time steps, respectively. φ is an activation function and, as always b_r denotes a learned bias term. Once we have r_t, we can learn y_t:

$$y_t = \text{softmax}(w^y \cdot r_t + b_y)$$

where w^y is a new set of weights that must be learned. While training, we look to minimize some loss function. While loss functions are somewhat application specific, one of the most common is cross-entropy.

A common technique used to intuit about the workings of a recurrent neural network is to "unroll" the network so that it looks more like a feedforward network with one layer per time step and weights shared across said time steps. Figure 3.1 shows how a simple RNN can be viewed as a feedforward network through the process of unfolding. Clearly this unrolled network will contain no cycles, which is significant in that it allows us to use a modified version of backpropagation. Called *Backpropagation through time* (BPTT), it is nearly identical to the version we already considered, except that we must deal with the fact that the weights are still shared across time steps. To address this, we average the shared weights after each is updated.

The main issue faced by BPTT is slow covergence. Due to our unrolling, the network will essentially grow after each iteration. Backpropagation and other training algorithms are computationally expensive, limiting the complexity of the networks we can utilize.

3.2 Liquid State Machines

In order to address some of the computational intractability of RNNs, Wolfgang Maass proposed the Liquid State Machine (LSM) in 2001 [30]. Largely inspired by discoveries in neuroscience, the pattern and distribution of connections between neurons in LSMs are biologically inspired. These connections can be recurrent, bidirectional, and in general have less rigid structure than in other network architectures. The result is a sort of "soup" of neurons all mixed together. Furthermore, the neurons themselves are biologically-

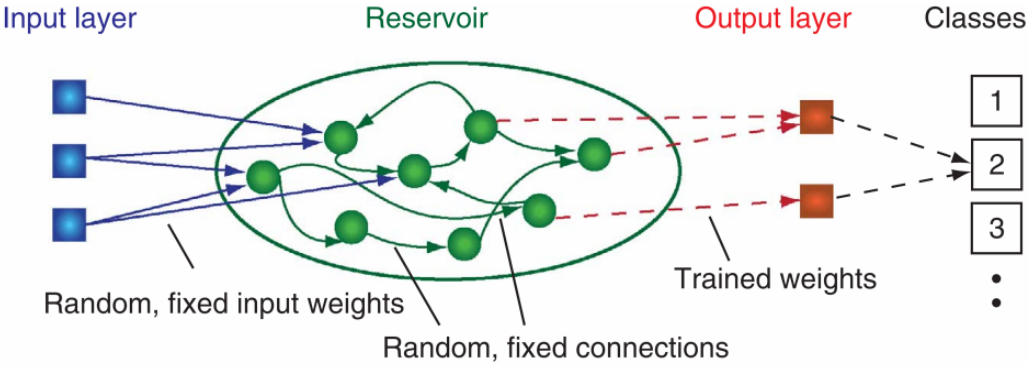

Figure 3.2: Model of a simple reservoir network. Image taken from [1].

inspired. LSMs use what is called a *spiking* neuron. Spiking neurons model neurons in the brain, where potential build up until it reaches some threshold, and then fires along outbound edges, using a trail of *activity bursts*. This technique is powerful in that it can approximate a wide variety of functions. The downside is that developing an effective LSM requires a large engineering effort to construct the network, as simulating biological neuron topologies is a dynamical system in and of itself, making the use of LSMs computationally expensive.

3.3 Echo State Networks

Developed by Herbert Jaeger [20] contemporaneously with LSMs, the Echo State Network (ESN) is a similar development in that it utilizes the same "soup"-like population of neurons. Jaeger called the soup a *dynamical reservoir*, and it consisted of a large cluster of randomly populated neurons

with many interconnections and recurrences. The reservoir was connected to a final output layer similar to that of more familiar networks. Jaeger showed that ESNs with a sufficient number of neurons and connections in the reservoir could achieve high accuracy on classification tasks even when only the output layer was trained. Unlike LSMs, ESN neurons are generally more typical sigmoid neurons. Because the network is randomly populated and the neurons are more simple, ESNs offer more computational efficiency compared to LSMs.

3.4 Reservoir Networks

Both LSMs and ESNs fall under the umbrella of what is today called *reservoir computing*. A typical reservoir network consists of an input layer, a reservoir, and an output layer. The difference between a reservoir network and a typical RNN is that training error only prompts an update to the weights of the neurons in the output layer. All the weights of the nodes in the reservoir are set ahead of time and are not updated during training.

The architecture of the reservoir itself is application dependent, but some common choices include a randomly populated set of neurons and connections, a biologically-inspired architecture, and even fully connected reservoirs. Reservoir architecture is an active area of research. While it remains to be determined what properties give a reservoir network the most learning potential, there are some properties that researchers have found to be advantageous in training reservoir networks.

The first such property is the *echo state property*. Postulated by Jaeger

in his proposal of the ESN, the echo state property mandates that the influence of a current state on future states diminish over time. Like sound echoing off the walls of a cavern, every time data bounces off a neuron in the network, it should not persist or amplify, but rather die off slowly. It is important, however, that it does not die off too quickly because otherwise we will lose the important ability of RNNs to have memory of the inputs. One common test to verify that memory capacity is maintained is to test if the network can recreate its input.

The second important property of reservoir networks is the separability property. While perhaps obvious, it has been shown that reservoir models for which different inputs produce different outputs can approximate a wider variety of functions. Furthermore, this also gives us intuition on how the size of our network influences its predictive power. Because a larger network can have a wider variety of activations, it has higher separability power.

Apart from a larger number of neurons yielding a more powerful network, the optimal architecture of reservoirs is largely unknown. While many researchers have postulated that a fully-connected reservoir will produce the best results, others have countered that sparse reservoirs can achieve the same accuracy on classification tasks. Furthermore, while some have argued that a randomly populated reservoir cannot possibly be optimal and have shown wide variance in the performance of random reservoirs, no hand-crafted topology has been able to significantly improve on the results yielded by random networks [13].

3.5 Applications of Reservoir Computing

Reservoir computing is still relatively new and has been less popular than some of the other forms of deep learning which have become widely studied in the last decade. Nevertheless, the reservoir paradigm has already been applied to many areas of research. Because of the developments of Maass, a neuroscientist by trade, in LSMs, reservoir computing has been largely studied by neuroscientists who see the paradigm as being similar to the topology of neurons in the brain. Researchers have used reservoir computing as a biological explanation for tasks like natural language learning [15, 16, 22], visual image processing [22, 21], and a variety of other tasks that are interesting from the perspective of modeling the way the human brain solves them. In addition, reservoir computing has also been used for more engineering-focused statistical learning [2, 29, 39] because of its ability to attain good results with less overall training time.

Chapter 4

Generative Models

Generative models are currently one of the most popular sub-areas of deep learning. As opposed to many familiar learning tasks that seek to learn how to classify/regress novel examples by training on existing examples, generative models learn instead how to *create* the novel examples themselves. Hopefully it is clear that this is a considerably more complicated task, but recently researchers have found success with generative models. We will look specifically at two categories of generative models: Generative Adversarial Networks (GANs) and Autoregressive models.

4.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks, originally proposed in 2014 by a group out of the University of Montreal [12], are a method for generating examples that replicate as some class as closely as possible by framing the task as a sort of competition between two networks. The first network, the generator, tries to create the synthetic instances of the class. The second, the discriminator, is a classifier that tries to guess whether a given example is synthetic, i.e. created by the generator, or a real example. When the discrim-

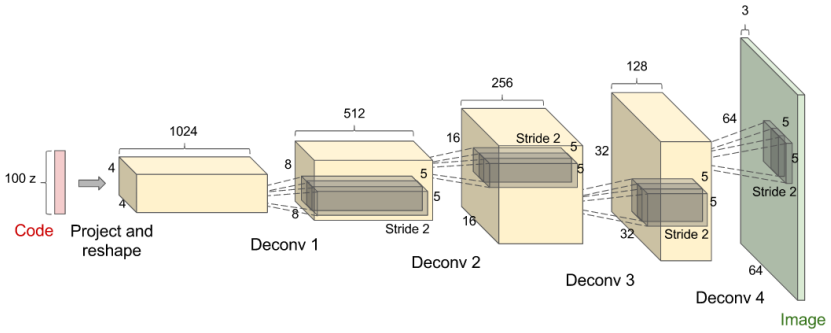

Figure 4.1: Overview of the DCGAN gnerator. The network (in yellow) is made up of standard convolutional neural network components, such as deconvolutional layers (reverse of convolutional layers), fully connected layers, etc. Taken from OpenAI.com.

inator finds a difference that allows it to pin down an example as synthetic, the generator adjusts its parameters to attempt to rid subsequent examples of the traits that gave the previous ones away. We backpropagate through both networks so that each is improving in its task. In theory, after going back and forth in this game with the discriminator, the generator will be able to produce samples that the discriminator cannot classify as being synthetic or real.

A recent and popular generator model is one proposed by Alec Radford, Luke Metz, Soumith Chintala in 2014 called DCGAN [40]. It is a feedforward network that takes as input 100 randomly generated numbers and outputs an image. The network consists primarily of deconvolutional layers that allow us to up-sample the 100 random numbers in order to create a 64×64 pixel image. The output layer uses a *tanh* activation and all others use *ReLU*.

Figure 4.1 shows a diagram of the model architecture. Remarkably, a simple vector of length 100 populated with random noise is transformed to create novel examples that mirror natural images.

4.2 Autoregressive Models

An autoregressive model trains a network that tries to model the conditional probability distribution for each stage of creating an example. In Natural Language Processing, we could look to generate new sentences [23]. Taking an autoregressive approach to this problem amounts to predicting the likelihood of a word given the words that proceeded it based on the frequencies observed from an existing corpus. In image generation like PixelRNN [38], this amounts to generating a first pixel and then recursively generating all the subsequent pixels based on the pixels near them in two-dimensional space.

PixelRNN uses softmax loss for training, making it significantly simpler to train than GANs. They also achieve the highest log-likelihood for plausibility of what was generated being an authentic image. The main downside of Autoregressive models is that they are computationally inefficient compared to the existing alternatives. This inefficiency stems from the fact that these networks build their samples piece by piece. In the case of images, samples are generated pixel-by-pixel, causing the run-time to be higher than other methods.

Chapter 5

GAR: A New Model

In this section we will motivate and develop a new network called a Generative Adversarial Reservoir (GAR). As the name might suggest, GAR is a combination of many of the concepts covered in sections 2-4. We will use GAR for prediction of natural videos.

5.1 Motivation

One of the most significant challenges posed by predicting video frames is the inherent high dimensionality. The resulting computational complexity of natural video prediction can be prohibitive in many useful applications. Extrapolating frames in a video has important use cases, especially when viewed from the perspective of programming autonomous robotic agents. In order for an agent to perform a task well, it must be able to ingest and process information about its current state, predict a future state, and take an action that prepares it to achieve its goal. Many of these interactions, naturally, are governed by the laws of physics, which has led some [35] to learn these interactions using manually labeled datasets to predict physics. While promising, the value and feasibility of apprenticeship learning have been demonstrated [5]

and motivated. An effective approach for learning via unlabeled video examples would be especially beneficial for natural video prediction because of the nearly unlimited resources available.

5.2 Related Work

Video Prediction has become a popular area of research in the last 2-3 years, as applications for such methods become more and more relevant to ongoing problems in both academia and industry. For the sake of brevity, we introduce some of the most recent and influential in regards to our new model. In 2014 Ranzato et al. showed how optical flow broke down spectacularly for certain cases and proposed a baseline for prediction of natural videos using techniques akin to the recurrent methods used in language models. Mathieu et al. [31] showed that training a generative adversarial network with novel loss functions for video prediction could provide greatly improve the "sharpness" of predicted frames. Most recently, Finn et al. improved on long-range video prediction using convolution LSTMs [9].

5.3 Formulation of the Model

Given a sequence $X_1, X_2, ..., X_N$ of consecutive video frames, we want to train a system that is able to accurately generate subsequent frames $X_{N+1}, X_{N+2}, ...XN + t$ where t is the number of future frames we would like to predict. Furthermore, given a sequence $X_1, X_2, ..., X_N$ where X_n where $1 < n < N$ is missing, we would like to learn to "interpolate" the missing frame. This well-defined task

clearly requires no labeling information about what is happening in the videos.

Now consider a (GAN) consisting of a *generator*, G, and a *discriminator*, D. Typically, as in [12], novel images are generated by G initialized with random noise. D then estimates the probability that a given example was generated by G as opposed to coming from the dataset of real training examples. The networks are trained in parallel so that G learns to generate images that are difficult for D to classify and D learns to determine the source of an image.

Mathieu et. al modified the approach for natural video prediction. In their model, G is still given with the same task of generating novel images. D, on the other hand, also takes a sequence of frames from a video. D is trained to estimate the probability that the last t frames of the are real or generated by G. Because D uses temporal information to discriminate between real and synthetic videos, G learns to generate temporally coherent sequences. Furthermore, the random noise used in regular GANs is no longer necessary since the generator can condition on real frames from the video.

Consider (X, Y) to be a sample from the dataset of natural videos. X is a sequence of N real frames and Y is a sequence of $N + t$ real and generated frames. We train D and G in an alternating fashion, meaning we keep the weights of one network fixed and perform one SGD step on the other network.

In order to train G, the authors seek to minimize the adversarial loss given by

$$\mathcal{L}_{adv}^{G} = \sum_{k=1}^{N} L_{bce}(D_k(X_k, G_k(X_k)), 1)$$

where L_{bce} is the standard binary cross-entropy loss. The authors of [31] note that utilizing this loss function alone can lead to instability since as G attempts to "confuse" D, there is no guarantee that it will do so without making its samples Y' close to Y. Thus they combine the above loss function with the $\mathcal{L}_p$ loss. Therefore G tries to minimize $\lambda_{adv}\mathcal{L}_{adv}^{G} + \lambda_{\ell_p}\mathcal{L}_p$ where λ_{adv} and λ_{ℓ_p} are tunable parameters that weight the trade-off between the sharpness of the frames and the similarity to the ground truth images.

Training D then amounts to minimizing the loss function

$$\mathcal{L}_{adv}^{D} = \sum_{k=1}^{N} L_{bce}(D_k(X_k, Y_k), 1) + L_{bce}(D_k(X_k, Y_k'), 0)$$

where $Y' = G(X)$, the predicted frames from the generator.

The authors of [31] use the Image Gradient Difference Loss (GDL), formulated as

$$\mathcal{L}_{gdl}(Y', Y) = \sum_{i,j} \big| |Y_{i,j} - Y_{i-1,j}| - |Y'_{i,j} - Y'_{i-1,j}| \big|^{\alpha} + \big| |Y_{i,j-1} - Y_{i,j}| - |Y'_{i,j-1} - Y'_{i,j}| \big|^{\alpha}$$

where α is a parameter greater than or equal to 1. The merits of this loss function are detailed in [31], but the main point is that GDL penalises the gradient disparity between the ground truth and prediction by considering the neighbor pixel intensity differences, keeping training time relatively low while also making sure the predicted image most closely approximates the ground truth.

Finally, the authors of [31] use the following composition of loss functions to train G:

$$\mathcal{L}^G = \lambda_{adv}\mathcal{L}^G_{adv} + \lambda_{\ell_p}\mathcal{L}_p + \lambda_{gdl}\mathcal{L}_{gdl}.$$

In early 2016, Im et al. [19] proposed Generative Recurrent Adversarial Networks (GRAN), whose structure is similar to GANs except that the generator G consists of recurrent connections that accumulate updates at each time step. GRAN was used to generate novel images and showed improvement over traditional adversarial nets.

A GAR is a GRAN where the RNN underlying the generator is substituted with a randomly generated reservoir. Optimal reservoir architecture is an active area of research, and while some have argued that a randomly populated reservoir cannot possibly be optimal and have shown wide variance in the performance of random reservoirs, no hand-crafted topology has been able to significantly improve on the results yielded by random networks [13].

5.4 Model Architecture

Figure 5.1 shows the network architecture we employ for video prediction. The discriminator is identical to that of [31], while the generator is designed so that each layer in [31] is replaced by a reservoir whose output layer is of the same dimension as that layer. As before, the generator and discriminator alternate training using the previously described loss functions. The only difference is that training only occurs on the output layer.

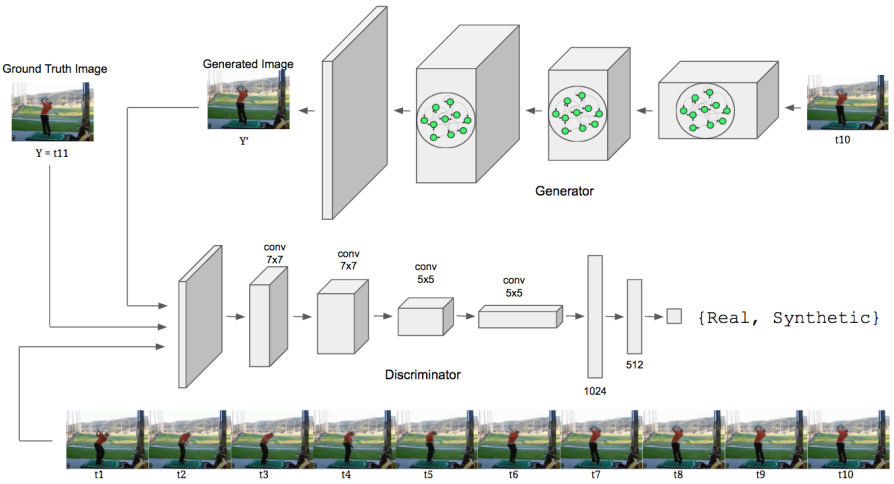

Figure 5.1: Network architecture diagram for training GAR using a context of 10 video frames. The generator outputs Y', an image representing the prediction for the next frame. The discriminator estimates the probability that a given image comes from the original dataset or the generator, conditioning on the prior frames.

5.5 Experiments

We now provide quantitative and qualitative results for evaluation of our network. We apply GAR as a fast video prediction method. Training was performed on Maverick, an HP/NVIDIA interactive visualization and data analytics system housed at the Texas Advanced Computing Center (TACC). Each node on Maverick Tesla K40 GPU and two Intel Xeon E5-2680 v2 Ivy Bridge CPUs running at 2.80 GHz and 1/4 TB of total memory.

Unlike [31] which conditions the discriminator network on four and eight frames, our setup enables us to use ten prior frames. We first test our model for generating one frame and then apply it recursively to generate

eight future frames. Clearly the more frames generated, the more difficult the problem becomes. Just as in [31], we apply this motion to a random 64×64 pixel patches that optical flow indicates as showing sufficient movement. These patches are normalized so that their values fall within $[-1, 1]$.

5.6 Training Details

We present images that were generated using the baseline model developed by Ranzato et al., images generated by non-adversarial (L1 and L2) loss, results achieved by conventional generative loss, and finally frames predicted by GAR.

We use the following values for the model parameters for GAR, mirroring those of [31]:

$$\lambda_{adv} = 0.05$$

$$\lambda_{\ell_p} = 1$$

$$\lambda_{gdl} = 1$$

$$\alpha = 1$$

$$\text{learning rate} = 0.04 \text{ decaying to } 0.005$$

$$\text{batch size:}10$$

The discriminator, D, uses ReLU non-linearities and 2×2 pooling after convolutions. The learning rate for D is 0.02.

5.7 Quantitative Evaluation

The metric we use to evaluate the similarity of our predicted frame, Y', to the true frame, Y is Peak Signal to Noise Ratio (PSNR) given by

$$\text{PSNR}(Y, Y') = 10 \cdot log_{10} \frac{max_{Y'}^2}{\frac{1}{N} \sum_{i=0}^{N} (Y_i - Y_i')^2}.$$

Here $max_{Y'}$ denotes the highest image intensity value possible. In keeping with tradition, we also report the Structual Similarity Index Measure (SSIM) developed by Wang et al. (2004). SSIM is a symmetric function given by:

$$\text{SSIM}(Y, Y') = \frac{(2\mu_Y \mu_{Y'} + c_1)(2\sigma_{YY'} + c_2)}{(\mu_Y^2 + \mu_{Y'}^2 + c_1)(\sigma_Y^2 + \sigma_{Y'}^2 + c_2)}$$

where μ_Y and $\mu_{Y'}$ denote the average value for Y and Y', respectively, σ_Y and $\sigma_{Y'}$ represent the variance for Y and Y', respectively, $\sigma_{Y,Y'}$ is the covariance between Y and Y' and c_1 and c_2 are stabilization constants.

5.8 Results

Table 5.1 shows a comparison of the results for 1st frame prediction on an unseen test set of 20% of UCF-101 videos. It is worth noting that these results differ somewhat from those reported in [31]. These minor discrepancies most likely arise from minor differences in Mathieu et al.'s official code, which is implemented in Torch, and the code for this work, which was implemented in TensorFlow. The Adv+GDL models were also not fine-tuned as they were

in [31]. Furthermore, we report results for Ranzato et al., but these are taken from averges of the examples reported from Mathieu et al., not an actual implementation of their method. It is included here because Ranzato et al. is seen as the baseline for video prediction, but the values reported likely do not represent the actual accuracy values that would be attained by their model.

First Frame Prediction Accuracy		
Technique	PSNR	SSIM
Optical Flow	31.4	**0.95**
L1	28.1	0.87
L2	26.8	0.85
Ranzato et al*	18.2	0.71
Mathieu et al.	**31.7**	0.92
GAR	30.1	0.90

Table 5.1: Results for prediction of first unseen frame.

At first glance these results do not appear to favor our method, as both accuracy metrics fell short of Mathieu et al. for first frame prediction. Figure 5.2 shows the training loss over time for the two methods. While Mathieu et al. converges to a lower overall loss, our method converges faster because of the use of reservoirs for training.

Figures 4 and 5 show qualitative examples comparing the results of Mathieu et al. and GAR with the ground truth.

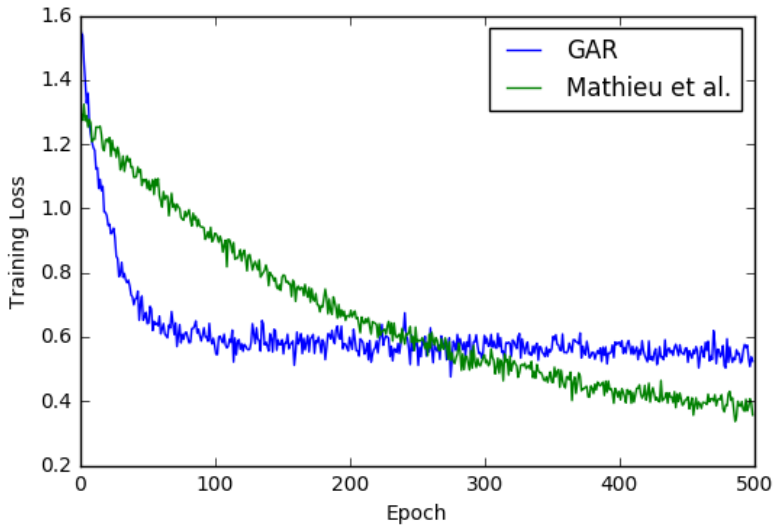

Figure 5.2: Comparing the training loss over time for GAR and method proposed by Mathieu et al.

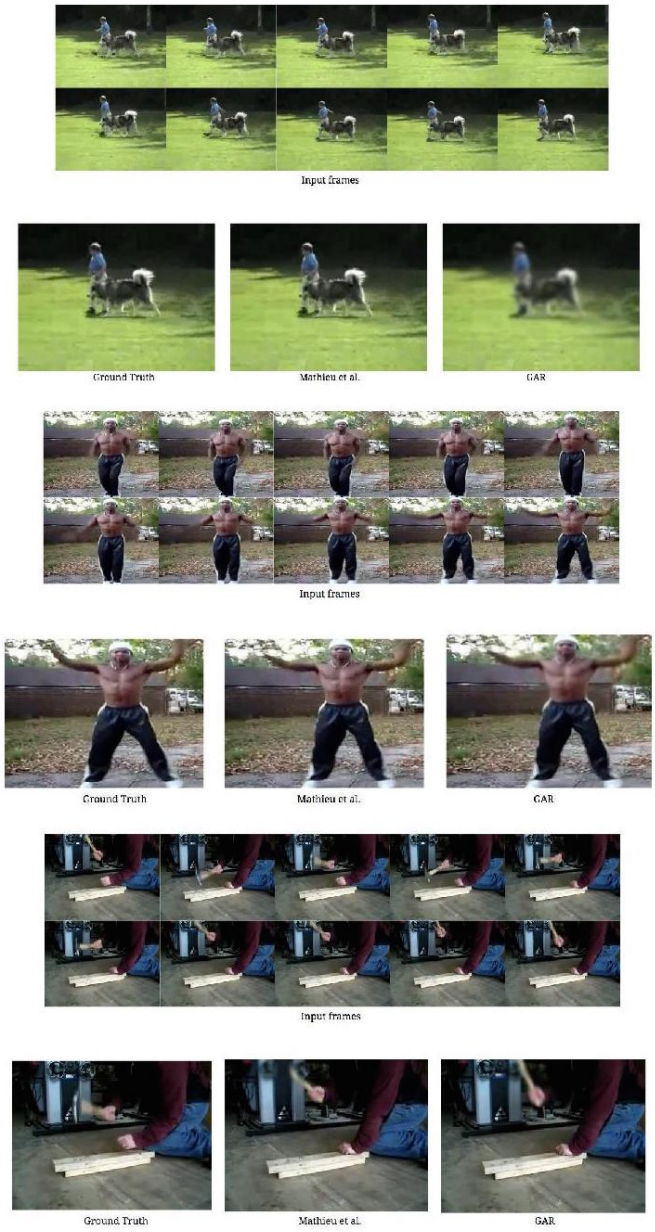

Figure 5.3: Results for predicting one frame of videos from UCF-101 (Part 1)

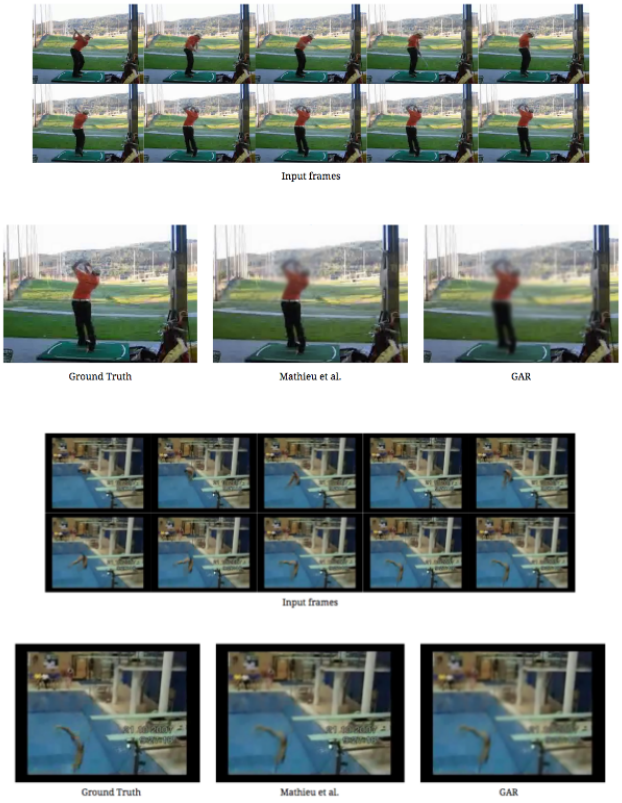

Figure 5.4: Results for predicting one frame of videos from UCF-101 (Part 2)

Chapter 6

Conclusion and Future Work

In this work, we showed that a GAN could be modified to train using the reservoir computing paradigm. We introduced the GAR, a network that trains adversarially but replaces the layers of the network introduced in [31] with randomly generated reservoirs. While this does not yield state-of-the-art results as did [31] for image sharpness or [9] for length of prediction, it did train notably quickly.

Future work on this topic would include exploring loss functions that would retain the sharpness resulting from adversarial training as in [31], retaining the ability to predict long sequences of video as in [9] and also retain the ability to train quickly. A network which accomplished all of those goals would be a powerful tool that could be used in many different applications.

Bibliography

[1] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer. *Information processing using a single dynamical node as complex system.* Nature communications, 2011.

[2] S. Basterrech, C. Fyfe, and G. (2011 Rubino. November). In *Self-organizing maps and scale-invariant maps in echo state networks*, pages 94–99. In Intelligent Systems Design and Applications (ISDA)11th International Conference on . IEEE, 2011.

[3] Y. Bengio, I. J. Goodfellow, and A. Deep Learning Courville. *Book in preparation for MIT Press (2015).* Disponvel em.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[5] M. Bogdanovic, D. Markovikj, M. Denil, and N. de Freitas. *Deep Apprenticeship Learning for Playing Video Games (Doctoral dissertation.* University of, Oxford, 2014.

[6] M. Campbell, A. J. Hoane, and F. H. Hsu. Deep blue. *Artificial intelligence*, 134(1):57–83, 2002.

[7] G. E. Dahl. *Deep learning approaches to problems in speech recognition.* computational chemistry, and natural language text processing (Doctoral dissertation, University of Toronto, 2015.

[8] G. E. Dahl, T. N. Sainath, and G. E. (2013 Hinton. May). In *Improving deep neural networks for LVCSR using rectified linear units and dropout. InIEEE International Conference on Acoustics*, pages 8609–8613. Speech and Signal Processing . IEEE, 2013.

[9] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. arxiv. preprint, arXiv:1605.07157, 2016.

[10] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[11] K. I. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, and S. Ozair. ... bengio, y. (2014). *Generative adversarial nets*, pages 2672–2680.

[13] F. Grezes. *Reservoir Computing (Doctoral dissertation.* The City University of, New York, 2014.

[14] S. Grossberg. Contour enhancement, short term memory, and constancies in reverberating neural networks. In *Studies of mind and brain*, pages 332–378. Springer, Netherlands, 1982.

[15] X. Hinaut and P. F. Dominey. Real-time parallel processing of grammatical structure in the fronto-striatal system: a recurrent network simulation study using reservoir computing. *PloS one*, 8:2, 2013.

[16] X. Hinaut, M. Petit, and P. F. (2012 Dominey. September). In *Online Language Learning to Perform and Describe Actions for Human-Robot Interaction*. In Post-Graduate Conference on Robotics and Development of Cognition (p. 59.

[17] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[18] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] D. J. Im, C. D. Kim, H. Jiang, and R. Memisevic. Generating images with recurrent adversarial networks. arxiv. preprint, arXiv:1602.05110, 2016.

[20] H. Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007.

[21] A. Jalalvand, F. Triefenbach, D. Verstraeten, and J. P. Martens. Connected digit recognition by means of reservoir computing. *In INTERSPEECH (pp.*, pages 1725–1728, 2011.

[22] A. L. Jouen, T. M. Ellmore, C. Madden, C. Pallier, P. F. Dominey, and J. Ventre-Dominey. *A common meaning system for language and visual images revealed by fMRI and DTI.* Society for Neuroscience, 2012.

[23] A. Karpathy. *Char-RNN: Multi-layer recurrent neural networks (LSTM.* GRU, RNN) for character-level language models in torch, 2015.

[24] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems (pp.*, pages 1097–1105, 2012.

[26] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[27] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324. 86(11, 1998.

[29] M. Lukoeviius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, Berlin, 2012.

[30] W. Maass. Liquid state machines: motivation, theory, and applications. *Computability in context: computation and logic in the real world*, pages 275–296, 2010.

[31] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. arxiv. preprint, arXiv:1511.05440, 2015.

[32] M. Minsky and S. Papert. Perceptrons, 1969.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. arxiv. preprint, arXiv:1312.5602, 2013.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, and M. G. Bellemare. ... petersen, s. (2015). *Human-level control through deep reinforcement learning*, 518(7540):529–533.

[35] R. Mottaghi, H. Bagherinezhad, M. Rastegari, and A. Farhadi. Newtonian image understanding: Unfolding the dynamics of objects in static images. arxiv. preprint, arXiv:1511.04048, 2015.

[36] A. B. Novikoff. On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata*, pages 615–622, Polytechnic Institute of Brooklyn, 1962. 12.

[37] M. Olazaran. A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659, 1996.

[38] A. V. D. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. arxiv. preprint, arXiv:1601.06759, 2016.

[39] M. Pfeiffer, B. Nessler, R. J. Douglas, and W. Maass. Reward-modulated hebbian learning of decision making. *Neural Computation*, 22(6):1399–1444, 2010.

[40] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arxiv. preprint, arXiv:1511.06434, 2015.

[41] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: a baseline for generative models of natural videos. arxiv. preprint, arXiv:1412.6604, 2014.

[42] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation (no. *ICS*-, 8506, 1985.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[45] S. Saha and G. P. S. Raghava. Prediction of continuous bâcell epitopes in an antigen using recurrent neural network. *Proteins: Structure, Function, and Bioinformatics*, 65(1):40–48, 2006.

[46] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, and Van Den Driessche. G., ... dieleman, s. (2016). *Mastering the game of Go with deep neural networks and tree search*, 529(7587):484–489.

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arxiv. preprint, arXiv:1409.1556, 2014.

[48] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. arxiv. preprint, arXiv:1212.0402, 2012.

[49] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, and D. Anguelov. ... rabinovich. In *A*, pages 1–9. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.

[50] D. S. Touretzky. *The mathematics of inheritance systems (Vol. 8)*. Morgan Kaufmann, 1986.

[51] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58:5, 1936.

[52] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio. Renet: A recurrent neural network based alternative to convolutional networks. arxiv. preprint, arXiv:1505.00393, 2015.

[53] P. Werbos. Backpropagation through time: what it does and how to do it. In *Proceedings of the IEEE*, 1990.

[54] P. J. (1988 Werbos. July). *Backpropagation: Past and future*, pages 343–353, 1988.

Vita

Jeremy James Hintz received his Bachelor of Science in Computer Science from The University of Texas at Austin. After graduation, Jeremy is going to work for Uber in Palo Alto, CA, applying the concepts of deep learning in order to create a transportation network as reliable as running water.

During his studies, Jeremy interned for Uber in San Francisco, CA. He developed a predictive capacity model of Uber's most heavily queried real-time services that is used to intelligently provision servers for future traffic. Previously at RetailMeNot in Austin, TX, Jeremy built a recommender system using Spark MLLib that is used throughout the company. Previously at Samsung Austin Semiconductor, he developed a new methodology for processing streaming data to cut down on overall manufacturing stop-loss.

For an ongoing list of Jeremy's projects in deep learning as well as other fields, go to http://jjhintz.com. To contact Jeremy directly, email at jeremy.hintz@gmail.com or call/text (214) 883-1547.

This report was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.